
Enterprise AI Hub Nexus

Technical Documentation for Developers

Document type:	Engineering Reference
Version:	1.0
Date:	March 2026
Audience:	Backend & Frontend Developers, DevOps
Product:	Enterprise AI Hub Nexus
Company:	OLIVER Agency

Table of Contents

1. Architecture Overview
2. Technology Stack
3. Repository Structure
4. Authentication Flow (PKCE + JWT)
5. Backend — FastAPI Application
6. RAG Pipeline
7. LLM Factory
8. Document Processing Pipeline
9. Cloud Run Document Processor
10. Frontend — Next.js Application
11. Deployment & Infrastructure
12. Database Migrations
13. Environment Variables Reference
14. API Reference

1. Architecture Overview

Enterprise AI Hub Nexus is a multi-tier, cloud-hybrid AI application. The frontend is a static Next.js SPA served by Apache. The backend is a FastAPI application running in Docker Compose on a Google Compute Engine VM. Heavy document processing (text extraction and chunking) is offloaded to Google Cloud Run. Vector search is handled by a self-hosted Qdrant instance. Authentication is delegated to Azure AD (Entra ID) using the PKCE OAuth flow.

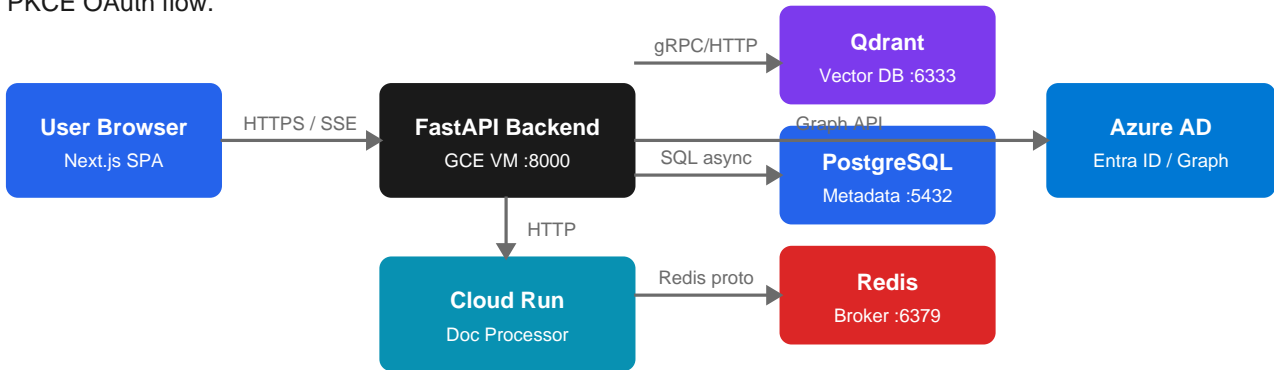


Figure 1 — High-level architecture diagram

2. Technology Stack

Layer	Technology	Version / Notes
Frontend	Next.js	14 (App Router, static export, basePath: /nexus)
Frontend	React + TypeScript	18 / TS 5
Frontend	Tailwind CSS + shadcn/ui	Component design system
Frontend	Zustand	Client state management
Backend	FastAPI + Python	0.115+ / Python 3.11+
Backend	SQLAlchemy (async)	2.x with asyncpg driver
Backend	Alembic	Database schema migrations
Backend	Celery + Redis	Async task queue and scheduler
AI — RAG	OpenAI GPT-5	gpt-5.2 — RAG question answering
AI — Assistant	Anthropic Claude Sonnet	claude-sonnet-4-6 — agentic tool loop
AI — Reranking	Anthropic Claude Haiku	claude-haiku-4-5 — rerank, summaries, query expansion
AI — Summary/Plan	Google Gemini	gemini-3.1-pro-preview — summarisation, planning
AI — Embeddings	OpenAI Embeddings	text-embedding-3-large (3072 dimensions)
Vector DB	Qdrant	1.12.x self-hosted on GCE VM

Layer	Technology	Version / Notes
Relational DB	PostgreSQL	15
Cloud Processing	Google Cloud Run	Document processor microservice
Infrastructure	GCE VM (n2d-standard-4)	Backend + Qdrant + Postgres + Redis
Auth	Azure AD / Entra ID	PKCE SPA flow — no client_secret
Auth	Microsoft Graph API	v1.0 — user profile + M365 tools
Web server	Apache 2.4	Reverse proxy + static file serving
Containers	Docker Compose	docker-compose.prod.yml

3. Repository Structure

The monorepo contains both the frontend and backend. Key directories are described below:

Path	Description
backend/app/api/v1/endpoints/	FastAPI route handlers: auth, chat, knowledge, users, config
backend/app/core/	Core services: DocumentProcessor, LLMFactory, CloudRunClient, web_scraper
backend/app/rag/	RAG retriever, query expansion, LLM reranker
backend/app/tools/	Personal assistant tools (email, calendar, OneDrive, SharePoint)
backend/app/models/	SQLAlchemy ORM models (User, KnowledgeDocument, KnowledgeSource, etc.)
backend/app/schemas/	Pydantic request/response schemas
backend/app/config.py	Settings via pydantic-settings (reads env vars)
backend/app/database.py	Async SQLAlchemy engine and session factory (AsyncSessionLocal)
backend/alembic/versions/	14 Alembic migration scripts
backend/cloud_run_processor/	Cloud Run microservice: text extraction + chunking only
frontend/app/	Next.js App Router pages (admin/, auth/callback/, chat/)
frontend/components/	React components: admin/, auth/, chat/, ui/ (shadcn)
frontend/lib/	API client, microsoft-oauth.ts (PKCE), utils
frontend/store/	Zustand stores: useAuthStore, useChatStore
frontend/types/	TypeScript type definitions
docker-compose.prod.yml	Production Docker Compose (backend, celery, redis, qdrant)
docker-compose.local.yml	Local dev (DB on 5433, Redis on 6380, backend on 1222)
deploy.sh	Full deploy script: git pull, build frontend, restart Docker

4. Authentication Flow (PKCE + JWT)

The platform uses OAuth 2.0 PKCE — suitable for Single-Page Applications (no client_secret needed). The backend never calls the Microsoft token endpoint; it only validates the MS access token against Graph API.

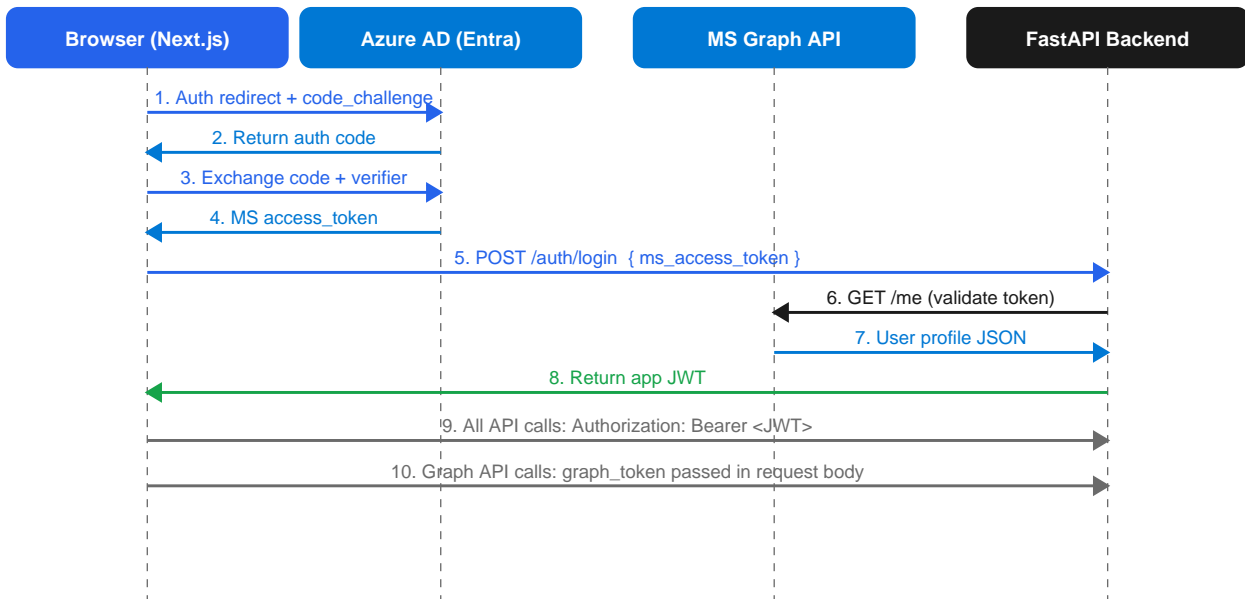


Figure 2 — PKCE OAuth + app JWT authentication sequence

4.1 Key Files

- **frontend/lib/microsoft-oauth.ts** — PKCE code_verifier/challenge generation; exchangeCodeForToken() calls Microsoft directly from the browser.
- **frontend/app/auth/callback/page.tsx** — receives OAuth code, calls exchangeCodeForToken(), then POSTs MS access_token to backend /auth/login.
- **backend/app/api/v1/endpoints/auth.py** — receives ms_access_token, calls Graph /me to validate it, provisions user in DB, returns signed JWT.
- **backend/app/schemas/auth.py** — LoginRequest schema: { ms_access_token: str }.
- **frontend/components/auth/protected-route.tsx** — auth guard with SSR hydration tracking.

4.2 JWT Configuration

- Algorithm: HS256. Secret: SECRET_KEY env var (32+ random bytes).
- Token lifetime: 8 hours (configurable via ACCESS_TOKEN_EXPIRE_MINUTES).
- Payload: { sub: user_id, email, role, department_id, region_code }.
- Validated on every request via FastAPI Depends(get_current_user).

4.3 Pending: GroupMember.Read.All

Note: The GroupMember.Read.All scope requires admin consent and has been temporarily removed from the OAuth request. Once admin grants consent, re-add it to microsoft-oauth.ts and auth.py to enable group-based department assignment.

5. Backend — FastAPI Application

5.1 Application Structure

```
# main.py — router registration
app.include_router(auth_router, prefix='/api/v1/auth')
app.include_router(chat_router, prefix='/api/v1/chat')
app.include_router(knowledge_router, prefix='/api/v1/admin/knowledge')
```

```
app.include_router(users_router, prefix='/api/v1/admin/users')
app.include_router(config_router, prefix='/api/v1/admin/config')
app.include_router(departments_router, prefix='/api/v1/admin/departments')
```

5.2 Database Session Pattern

Request-scoped sessions via Dependency Injection:

```
# Per-request session (HTTP handlers) async def get_db() -> AsyncGenerator[AsyncSession,
None]: async with AsyncSessionLocal() as session: yield session # Independent session
(background tasks) # IMPORTANT: always use own session in BackgroundTasks # to avoid
StaleDataError from long-lived request sessions async def _process_doc_background(doc_id,
...): async with AsyncSessionLocal() as db: doc = await db.get(KnowledgeDocument, doc_id)
...
```

5.3 Key Endpoints

Method	Path	Auth	Description
POST	/api/v1/auth/login	Public	Exchange MS token → app JWT
POST	/api/v1/chat/stream	User	SSE streaming RAG/assistant chat
GET	/api/v1/admin/knowledge/documents	Admin	List all documents (limit 1000)
POST	/api/v1/admin/knowledge/upload	Admin	Upload + background-queue document
POST	/api/v1/admin/knowledge/scrape	Admin	Scrape URL and index
POST	/api/v1/admin/knowledge/stats	Admin	Doc counts + Qdrant vector count
POST	/api/v1/admin/knowledge/documents/{id}/reprocess	Admin	Re-queue single document
POST	/api/v1/admin/knowledge/documents/bulk-delete	Admin	Delete multiple docs + vectors
POST	/api/v1/admin/knowledge/reindex-all	Admin	Re-queue all completed docs
DELETE	/api/v1/admin/knowledge/documents/{id}	Admin	Delete doc + Qdrant vectors
GET	/api/v1/admin/users	Super Admin	List all users
GET	/api/v1/admin/departments	Admin	List departments

5.4 Background Processing Pattern

Document uploads return **202 Accepted** immediately. Processing runs via FastAPI BackgroundTasks to avoid blocking HTTP and prevent StaleDataError:

```
@router.post('/upload', status_code=202) async def upload_document( file: UploadFile,
background_tasks: BackgroundTasks, db: AsyncSession = Depends(get_db), ): # 1. SHA-256
dedup check # 2. Save file to UPLOAD_DIR # 3. Create DB record with status=PENDING # 4.
Queue background task (own session) background_tasks.add_task( _process_doc_background,
doc_id=doc.id, file_path=saved_path, ... ) return {"id": doc.id, "status": "pending"}
```

6. RAG Pipeline

Implemented in **backend/app/rag/retriever.py**. The pipeline has four stages designed to maximise retrieval quality across multilingual and UK/US terminology variations:



Figure 3 — RAG pipeline: query expansion → embed → search → rerank → answer

6.1 Query Expansion

Before searching, Claude Haiku expands the user query into 3 variants to improve recall across languages and terminology variants:

- **Variant 1:** Normalised / translated version of the original query.
- **Variant 2:** UK English terminology (annual leave, holiday entitlement, redundancy, notice period).
- **Variant 3:** US English terminology (vacation, PTO, layoff, two weeks notice).

```
# retriever.py - _expand_query() prompt = ( f'Original query: {query}\n' 'Generate 3
search variants as JSON array:\n' '1. Translated/normalised version\n' '2. UK English
terminology (annual leave, holiday...)\n' '3. US English terminology (vacation,
PTO...)\n' 'Return ONLY the JSON array.' )
```

6.2 Parallel Embedding & Search

All 3 query variants are embedded concurrently with `asyncio.gather`, then searched in Qdrant in parallel. Results are merged and deduplicated by point ID (highest score kept), yielding up to `top_k * 2 * 3` candidates before reranking.

```
# Embed all variants in parallel embeddings = await asyncio.gather(
*[self._embeddings.aembed_query(q) for q in query_variants] ) # Search all variants in
parallel results = await asyncio.gather( *[self._search_qdrant(emb, filters, top_k) for
emb in embeddings] ) # Merge + dedup by point ID seen = {} for batch in results: for hit
in batch: if hit.id not in seen or hit.score > seen[hit.id].score: seen[hit.id] = hit
```

6.3 LLM Reranking

The merged candidate list (up to 60 chunks) is scored by Claude Haiku on a 0–10 relevance scale. Each chunk is scored independently; the top-5 by score are passed to the answer LLM. This replaces binary yes/no grading with a much higher-precision ranking.

```
# retriever.py - grade_documents() # Called once per chunk with (query, chunk_text)
prompt = ( f'Question: {query}\n'f'Document chunk: {text}\n' 'Rate relevance 0-10.
Respond with ONLY a number.' ) score = int(await LLMFactory.generate_completion('rerank',
...)) # Sort descending, take top 5 ranked = sorted(chunks, key=lambda x: x.score,
reverse=True)[:5]
```

6.4 Qdrant Filters

Searches are filtered based on user context:

```
must_conditions = [ FieldCondition(key='is_active', match=MatchValue(value=True)), ] if
department_id: must_conditions.append( FieldCondition(key='department_id',
match=MatchValue(value=department_id)) ) if region_code: must_conditions.append(
FieldCondition(key='region_code', match=MatchValue(value=region_code)) )
```

7. LLM Factory

backend/app/core/llm.py — centralised factory for all LLM calls. Supports streaming, non-streaming, and agentic tool-calling loops.

Mode	Provider	Model	Use case
rag	OpenAI	gpt-5.2	RAG question answering (streaming)
assistant	Anthropic	claude-sonnet-4-6	Personal assistant / agentic tool loop
rerank	Anthropic	claude-haiku-4-5	Reranking, doc summaries, query expansion
summary	Google	gemini-3.1-pro-preview	Document summarisation
planning	Google	gemini-3.1-pro-preview	Task planning
transcript	Google	gemini-3.1-pro-preview	Audio/video transcript processing
translation	Anthropic	claude-sonnet-4-6	Content translation

7.1 Available Methods

- **get_llm(mode)** — returns a LangChain ChatModel instance (or Google dict).
- **stream_completion(mode, messages)** — async generator yielding text tokens via SSE.
- **generate_completion(mode, messages)** — single non-streaming response string.
- **stream_with_tools(mode, messages, tools, tool_context)** — agentic loop: stream LLM, execute tool calls in parallel, feed results back, repeat up to max_rounds=5.

7.2 Agentic Tool Loop

The tool-calling loop (used in Personal Assistant mode):

```
for _round in range(max_rounds): # 1. Stream LLM response, collect tool_calls async for
chunk in llm_with_tools.astream(messages): yield {'token': chunk.content} collect
tool_calls from chunk... if not tool_calls: return # LLM gave final answer # 2. Check
consent (graph_token required for M365 tools) # 3. Execute all tools in parallel results
= await asyncio.gather(*[_execute_one(tc) for tc in executable]) # 4. Append ToolMessage
results to history # 5. Loop — LLM sees results and may call more tools
```

8. Document Processing Pipeline

Implemented in **backend/app/core/document_processor.py**. Uses a two-phase design to separate Cloud Run (extraction) from backend VM (embedding).

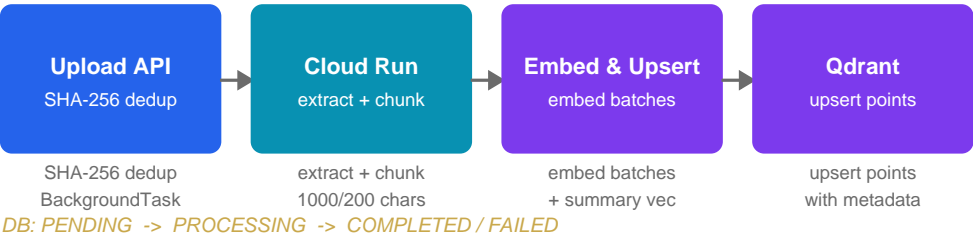


Figure 4 — Document ingestion pipeline

8.1 Two-Phase Design

- **extract_and_chunk(file_bytes, file_name, file_type)** — runs on Cloud Run. No Qdrant or OpenAI required. Returns list of text chunks.

- **embed_and_upsert(chunks, sharepoint_id, ...)** — runs on backend VM. Embeds in parallel batches of 100, generates document summary, upserts to Qdrant.
- **process_document()** — convenience wrapper that calls both phases locally (used when CLOUD_RUN_PROCESSOR_URL is not set).

8.2 Text Extraction

File type	Extractor	Notes
PDF (text-based)	MarkItDown	Fast, supports tables and multi-column layouts
PDF (scanned)	LlamaParse (cloud)	OCR via LlamaParse API; falls back to MarkItDown on error
DOCX / DOC	MarkItDown	Preserves headings, tables, lists
XLSX / XLS	MarkItDown	All sheets; cells separated by tabs
PPTX / PPT	MarkItDown	Slide text and speaker notes
TXT / CSV	Direct decode	UTF-8 with error replacement
Web page	trafilatura	Main content extraction; max 500 KB; timeout 30s

8.3 Contextual Chunks

Each chunk is prefixed with the document title before embedding to improve retrieval relevance. The original text is stored separately in the payload:

```
doc_title = file_name.rsplit('.', 1)[0] # Contextualised for embedding contextualized =
f'[{doc_title}]\n\n{chunk}' # Original stored in Qdrant payload for display payload =
{..., 'text': chunk, 'chunk_type': 'chunk'}
```

8.4 Document Summary Vector

After embedding, Claude Haiku generates a 3–4 sentence summary of the document. This summary is embedded as an additional Qdrant vector (chunk_type='summary', chunk_index=-1), improving discovery of documents by topic even when specific keywords are absent.

8.5 Deduplication

On upload a SHA-256 hash of the file bytes is computed. If a document with the same hash already exists in the DB, the upload returns 409 Conflict. This prevents re-indexing identical files regardless of filename changes.

8.6 Qdrant Point Schema

Field	Type	Index	Description
id	UUID string	—	Unique point ID per chunk
vector	float[3072]	HNSW	text-embedding-3-large embedding
sharepoint_id	string	keyword	MS Graph item ID (used for delete/update)
source_id	UUID	keyword	Knowledge source FK
file_name	string	—	Original filename
file_url	string	—	SharePoint / OneDrive URL
file_type	string	keyword	pdf, docx, xlsx, pptx, txt, url

Field	Type	Index	Description
department_id	UUID null	keyword	Department scope filter
region_code	string null	keyword	UK, US, APAC, EMEA etc.
chunk_index	integer	—	Position in doc (-1 = summary)
chunk_type	string	keyword	chunk summary
text	string	—	Original chunk text (not embedded text)
total_chunks	integer	—	Total chunks in parent document
is_active	boolean	bool	Soft-delete flag
author	string null	—	Document author from metadata
last_modified	ISO datetime	—	Document last modified
indexed_at	ISO datetime	—	Qdrant upsert timestamp

9. Cloud Run Document Processor

The Cloud Run microservice lives in **backend/cloud_run_processor/**. It handles the CPU-intensive document extraction and chunking, offloading this work from the backend VM to scale on demand and avoid timeouts.

9.1 Service Details

Property	Value
Service URL	https://nexus-processor-818629422283.europe-west1.run.app
Region	europe-west1 (Belgium)
Runtime	Python 3.11 (Cloud Run Gen 2)
Timeout	900 seconds (15 minutes)
Authentication	Google Identity Token (service account)
Trigger	HTTP POST from backend VM
Input	Multipart: file bytes + metadata
Output	JSON: { chunks: string[], chunk_count: int }

9.2 Request / Response Contract

```
# Backend sends to Cloud Run POST /process Content-Type: multipart/form-data
Authorization: Bearer Fields: file: file_name: 'Annual_Leave_Policy.pdf' file_type: 'pdf'
# Cloud Run response { 'chunks': ['chunk text 1', 'chunk text 2', ...], 'chunk_count': 42
}
```

9.3 CloudRunClient (backend/app/core/cloud_run_client.py)

The backend uses CloudRunClient to call the Cloud Run service. Key implementation details:

- TIMEOUT = 900.0 seconds (large documents like handbooks can take several minutes).
- `_get_identity_token()` fetches a Google identity token for service-to-service auth. This is a synchronous call wrapped in `asyncio.to_thread()` to avoid blocking the event loop.

- If `CLOUD_RUN_PROCESSOR_URL` env var is empty, the backend falls back to local `DocumentProcessor.extract_and_chunk()` — useful for local development without GCP.
- The client automatically refreshes the identity token on each request (tokens expire after 1 hour).

```
# cloud_run_client.py class CloudRunClient: TIMEOUT = 900.0 async def
process_document(self, file_bytes, file_name, file_type): token = await
asyncio.to_thread(self._get_identity_token) async with
httpx.AsyncClient(timeout=self.TIMEOUT) as client: resp = await client.post(
f'{CLOUD_RUN_URL}/process', headers={'Authorization': f'Bearer {token}'}, files={'file':
(file_name, file_bytes)}, data={'file_name': file_name, 'file_type': file_type}, ) return
resp.json()['chunks']
```

9.4 Deploying the Cloud Run Service

To deploy or redeploy the Cloud Run processor:

```
# From repo root cd backend/cloud_run_processor # Build and push container gcloud builds
submit --tag europe-west1-docker.pkg.dev/PROJECT_ID/nexus/processor # Deploy to Cloud Run
gcloud run deploy nexus-processor \ --image
europe-west1-docker.pkg.dev/PROJECT_ID/nexus/processor \ --region europe-west1 \
--platform managed \ --timeout 900 \ --memory 2Gi \ --cpu 2 \ --no-allow-unauthenticated
\ --service-account nexus-processor@PROJECT_ID.iam.gserviceaccount.com
```

9.5 IAM Setup

- The Cloud Run service must have **no-allow-unauthenticated** — only the backend service account can call it.
- The backend VM service account needs **roles/run.invoker** on the Cloud Run service.
- Grant via: `gcloud run services add-iam-policy-binding nexus-processor --member='serviceAccount:VM_SA@PROJECT.iam.gserviceaccount.com' --role='roles/run.invoker'`

9.6 Local Fallback

In local development (or when `CLOUD_RUN_PROCESSOR_URL` is not set), the backend calls `DocumentProcessor.extract_and_chunk()` directly in-process. This requires `MarkItDown` and optionally `LlamaParse` to be installed in the backend container. The `docker-compose.local.yml` is configured for this.

10. Frontend — Next.js Application

10.1 Build & Serving

- **Static export** (output: 'export') — no Node.js server at runtime.
- **basePath: /nexus, trailingSlash: true** — Apache serves from `/var/www/html/enterprise-ai-hub-nexus/`.
- Azure AD credentials (client ID, tenant ID) are hardcoded as defaults in `next.config.mjs` and `microsoft-oauth.ts` — no runtime environment variables needed at the client.
- Build: `npm ci && npm run build` — outputs to `out/` directory.

10.2 State Management

- **useAuthStore (Zustand)** — user object, JWT token, `login()` / `logout()` actions. Persisted to `sessionStorage`.
- **useChatStore (Zustand)** — messages array, current mode, streaming state, department/region selection.
- **ProtectedRoute** — wraps all authenticated pages. Uses a hydrated flag to prevent flash-of-unauthenticated-content during SSR→client handoff.

10.3 SSE Chat Streaming

Streaming uses `fetch` + `ReadableStream` over `text/event-stream`:

```
const response = await fetch('/api/v1/chat/stream', { method: 'POST', headers: {
  Authorization: `Bearer ${token}` }, body: JSON.stringify({ messages, mode, graph_token
}), }); const reader = response.body.getReader(); // Each SSE line: 'data: {json}\n\n' //
Event types: // { token: 'Hello' } - text chunk // { sources: [...] } - citations // {
tool_start: { name, args } } - tool invoked // { tool_result: { name, data } } - tool
completed // { done: true } - stream finished
```

10.4 Multi-file Upload Concurrency

KnowledgeUploader uses a worker pool pattern for 4-concurrent uploads:

```
const CONCURRENCY = 4; let idx = 0; async function worker() { while (true) { const i =
idx++; if (i >= pending.length) break; await uploadFile(pending[i]);
setProcessedCount(prev => prev + 1); } } await Promise.all(Array.from({ length:
CONCURRENCY }, worker));
```

10.5 Key Components

Component	File	Description
ChatInterface	components/chat/chat-interface.tsx	Main chat panel; SSE stream consumer; source citations
KnowledgeUploader	components/admin/knowledge-uploader.tsx	Drag-drop multi-file upload; SHA-256 dedup; 4-worker pool
SharePointBrowser	components/admin/sharepoint-browser.tsx	Browse & import from SharePoint document libraries
ProtectedRoute	components/auth/protected-route.tsx	Auth guard with hydration flag; redirects unauthenticated
IntegrationsTab	components/admin/integrations-tab.tsx	SharePoint source configuration (site URL, credentials)
UsersTab	components/admin/users-tab.tsx	User CRUD: invite, role assignment, department/region
DepartmentsTab	components/admin/departments-tab.tsx	Department management
AnalyticsTab	components/admin/analytics-tab.tsx	Usage analytics (queries, docs, users over time)
AdminPage	app/admin/page.tsx	Admin dashboard: knowledge base, users, LLM config, stats

11. Deployment & Infrastructure

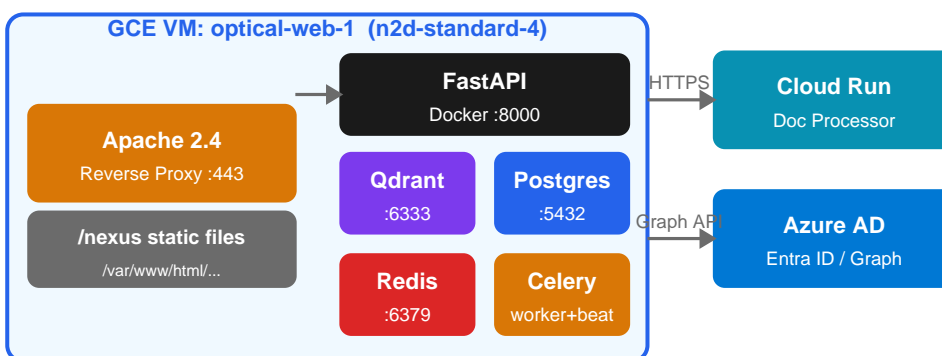


Figure 5 — Production infrastructure on GCE VM

11.1 Docker Compose Services

Service	Image	Port(s)	Description
backend	./Dockerfile	8000	FastAPI + uvicorn (4 workers)

Service	Image	Port(s)	Description
celery-worker	./Dockerfile	—	Celery worker; healthcheck: inspect ping
celery-beat	./Dockerfile	—	Celery periodic scheduler
redis	redis:7-alpine	6379	Message broker and cache
qdrant	qdrant/qdrant:v1.12.1	6333, 6334	Vector DB; data volume mounted

11.2 Apache Configuration

Apache acts as reverse proxy for the backend and serves the static frontend:

```
# /etc/apache2/sites-enabled/nexus.conf ServerName your-domain.com SSLEngine on # Reverse
proxy API to FastAPI container ProxyPass /api/v1/ http://localhost:8000/api/v1/
ProxyPassReverse /api/v1/ http://localhost:8000/api/v1/ # SSE: disable proxy buffering
for streaming ProxyPass http://localhost:8000/api/v1/chat/stream SetEnv proxy-nokeepalive
1 SetEnv proxy-sendchunked 1 # Static Next.js files Alias /nexus
/var/www/html/enterprise-ai-hub-nexus Options -Indexes AllowOverride All Require all
granted
```

11.3 Deploy Script (deploy.sh)

- git pull origin main — pull latest code.
- cd frontend && npm ci && npm run build — build static export.
- rsync -av out/ /var/www/html/enterprise-ai-hub-nexus/ — deploy frontend.
- docker-compose -f docker-compose.prod.yml up -d --build backend celery-worker — rebuild backend.
- docker exec backend alembic upgrade head — apply pending DB migrations.
- sudo systemctl reload apache2 — reload Apache config if changed.

12. Database Migrations

Managed with Alembic. 14 migration files in backend/alembic/versions/. All verified on a fresh PostgreSQL 15 database.

12.1 Common Commands

```
# Apply all pending migrations docker exec backend alembic upgrade head # Check current
revision docker exec backend alembic current # Generate a new migration from model
changes docker exec backend alembic revision --autogenerate -m 'add_column_xyz' #
Downgrade one step docker exec backend alembic downgrade -1 # View migration history
docker exec backend alembic history --verbose
```

12.2 Known Gotchas

asyncpg + sa.Enum: Do NOT use sa.Enum(create_type=False) — the flag is silently ignored by the asyncpg driver and the type will be created anyway, causing a 'type already exists' error. Let sa.Enum() inside op.create_table() handle creation naturally. Do NOT also add explicit DO \$\$ CREATE TYPE SQL in the same migration.

13. Environment Variables Reference

Variable	Required	Default	Description
DATABASE_URL	Yes	—	Async PostgreSQL DSN: postgresql+asyncpg://user:pass@host/db
REDIS_URL	Yes	—	Redis DSN: redis://localhost:6379/0
SECRET_KEY	Yes	—	JWT signing secret — min 32 random bytes
AZURE_CLIENT_ID	Yes	—	Azure AD app client ID (SPA platform)
AZURE_TENANT_ID	Yes	—	Azure AD tenant ID
OPENAI_API_KEY	Yes	—	OpenAI key — used for RAG (GPT-5) and embeddings
ANTHROPIC_API_KEY	Yes	—	Anthropic key — Claude Sonnet + Haiku
GOOGLE_API_KEY	No	—	Google Gemini key — summary / planning modes
QDRANT_URL	Yes	—	Qdrant HTTP URL: http://qdrant:6333
CLOUD_RUN_PROCESSOR_URL	No	—	Cloud Run URL; if empty uses local processor
LLMAPARSE_API_KEY	No	—	LlamaParse key for scanned PDF OCR
UPLOAD_DIR	No	/tmp/uploads	Directory to persist uploaded files on disk
CHUNK_SIZE	No	1000	Text chunk size in characters
CHUNK_OVERLAP	No	200	Overlap between consecutive chunks in characters
ACCESS_TOKEN_EXPIRE_MINUTES	No	480	JWT lifetime in minutes (default 8 hours)
BACKEND_CORS_ORIGINS	No	*	Comma-separated allowed CORS origins

14. API Reference

POST /api/v1/chat/stream

Streaming Server-Sent Events (SSE) endpoint. Returns text/event-stream.

```
# Request body { 'messages': [{ 'role': 'user', 'content': 'What is the leave policy?' }],
'mode': 'rag', // rag | assistant | general 'department_id': 'uuid|null', 'region_code':
'UK|US|null', 'graph_token': 'ms-token|null' // required for assistant mode } # Response
— one JSON per SSE line data: { 'token': 'The annual leave' } data: { 'token': ' policy
states...' } data: { 'sources': [{ 'file_name': 'UK_Leave.pdf', 'url': '...' }] } data:
{ 'done': true }
```

POST /api/v1/admin/knowledge/upload

Multipart form upload. Returns 202 immediately; processing is async.

```
# Request: multipart/form-data file: # document file (PDF, DOCX, XLSX, PPTX, TXT)
department_id: str|null region_code: str|null # Response 202 { 'id': 'uuid', 'file_name':
'doc.pdf', 'status': 'pending' } # Error 409 — duplicate file { 'detail': 'Duplicate file:
same content already exists as doc.pdf' }
```

POST /api/v1/admin/knowledge/stats

```
# Response { 'total_documents': 392, 'by_status': { 'completed': 385, 'processing': 0,
'pending': 2, 'failed': 5 }, 'zero_chunks_count': 3, 'total_vectors_qdrant': 47832 }
```

POST /api/v1/admin/knowledge/reindex-all

```
# Response { 'queued': 385, // documents queued for re-embedding 'skipped': 3 // docs  
whose file was not found on disk }
```

Enterprise AI Hub Nexus — Technical Documentation v1.0 | OLIVER Agency Engineering | March 2026 | Confidential